Amendments to the Specification:

Please amend the paragraph beginning on page 1, line 2 as follows:

This application is a Continuation-in-Part of co-pending U.S. Patent Application Serial No. 08/735,168, now U.S. Patent No. 6,754,656, entitled "System and Method For Selective Partition Locking," filed on October 22, 1996.

Please amend the paragraph beginning on page 2, line 10 as follows:

In a relational database system, contents of a database are represented as tables of data values. Each table corresponds to a relation. In a relational database, a table can be divided into partitions. Each partition contains a portion of the data in the table. A table may be divided into partitions based upon a range of values for a specified key. For example, in the language of the well-known DB2™ database system available from International Business Machines Corporation™, Armonk, New York, the syntax of a CREATE TABLESPACE statement includes a NUMPARTS clause that identifies the created table space as partitioned and sets the number of partitions. Partitions on a table in partitioned table space are characterized by a PART clause in a CREATE INDEX statement. Other forms of partitioning a table space are ~~is~~ possible. For example, table based partitioning can be used, where a partitioning index is not required.

Please amend the paragraph beginning on page 3, line 21 as follows:

With selective partition locking, only the partitions that are accessed by an application are locked. This allows highly concurrent access to separate portions of data in a table. For example, if a first application requires a shared (S) lock for reading data from partition A, while a second application requires an intent exclusive (IX) lock to update data in partition B, the incompatibility between the lock types will not result in delay of one application because the

table will not be wholly locked on behalf of the other application. Therefore, both applications can run concurrently, using the same table. Without selective partition locking, the ~~The~~ entire table would be locked and access for the applications would be serialized with respect to the table.

Please amend the paragraph beginning on page 5, line 2 as follows:

Our selective partition locking invention further improves ~~improved~~ performance by partition locking only if it cannot be determined whether the data has been committed. This further reduces the number of partition locks that need to be requested from a local resource lock manager, improving performance.

Please amend the paragraph beginning on page 5, line 11 as follows:

Other features and ~~advantage~~ advantages of the present invention will become apparent from the following more detailed description, taken in conjunction with the accompanying drawings, which illustrate, by way of example, the principles of the invention.

Please amend the paragraph beginning on page 5, line 16 as follows:

Fig. 1 is an illustration of data structures that exist in a relational database system following the creation and partitioning of a table using a series of SQL (structured query language) ~~statement~~ statements.

Please amend the paragraph beginning on page 6, line 8 as follows:

The preferred embodiment of our invention provides selective partition locking (SPL) by which a database management system (DBMS) acquires locks only on those partitions of a table

containing data that are ~~is~~ accessed by an application. According to our invention, selective partition locking leaves other partitions of that table available for concurrent access by other applications, regardless of the strength of the lock necessary to access the other partitions. This detailed description is intended to illustrate the principles underlying the invention and to signify the invention's reduction to practice. Manifestly, utilizing the teachings of this detailed description, corresponding systems and procedures may readily be conceived and reduced to practice for operating a database system.

Please amend the paragraph beginning on page 7, line 18 as follows:

The CREATE TABLESPACE statement creates table space TPIAA201, and identifies the table space as partitioned into eight partitions (NUMPARTS 8). The next statement creates a table TBIAA201 in the partitioned table space and defines the columns of the table. The third statement creates a partitioned index IPIAA201 on the table and defines the range of the partitioning key (LASTNAME) for each partition. The result is ~~illustrate din~~ illustrated in Fig. 1 which depicts the table 100 with eight partitions, of which three are shown 102, 103, and 104. The index 105 includes eight partitions, of which three are illustrated 106, 107 and 108. Each index partition indexes to a respective table partition and defines the range of values in that partition for the defined partitioning key. For example, the index partition 107 indexes to last names in which the beginning letter extends from D to F. Thus, the record 109 stored as a row in partition 103 is within the key range for that partition, since the value in the LASTNAME field is "Fuller".

Please amend the paragraph beginning on page 8, line 13 as follows:

A central electronic complex (CEC) 200 includes a general purpose programmed digital

computer on which programs that form a database management system (DBMS) 201 execute to provide users 204, 206 with access to a stored database 202. In the preferred embodiment, the database is a relational database whose contents are organized by relations that are represented as tables of data values, such as the table 203. A user 204 dispatches a request for access to the database 202 by way of an application 205 that is conventionally coupled to the DBMS 201. Other users such as the user 206 access the database 202 by way of the same or other respective application, such as the application 207. User requests ~~request~~ are received by a relational data system (RDS) component 210 of the DBMS 201. The RDS 210 decides what data stored at ~~a~~ the database 202 is required to satisfy a user request and requests access to the data on behalf of the application from which the request was received. A data manager (DM) 212 receives requests for access to records from the RDS 210. Before the DM 212 retrieves data requested by the RDS 210, ~~it~~ the DM 212 submits a request for a lock on the data to a resource lock manager (RLM) 214. The RLM 214 provides a locking service to the DBMS 201, managing locks on data in the database 202.

Please amend the paragraph beginning on page 9, line 13 as follows:

In the data-sharing environment, other CEC's such as the CEC 220 access contents of the database 202, with concurrency between the CEC's 200 and 220 being managed by a global lock manager (GLM) 222. In the data-sharing environment, the GLM 222 includes programming logic that supports hierarchical locking. This existing support is used in a novel way by the DM 212 with SPL 215 to achieve finer granularity in multi-CEC ~~concurrently~~ concurrency, i.e. partition-level contention as an alternate to table space contention.

Please amend the paragraph beginning on page 9, line 19 as follows:

Fig. 3 depicts the process by which selective partition locking is implemented in our invention. Fig. 3 contains references to Tables I-VII which are pseudocode representations of one or more computer programs (software) and of functions of the DM 212 that are performed by the selective partition locking component 215. In practice these functions may be performed by a programmed general purpose digital computer, for example, the DB2 system for the MVS/ESA operating system that executes on a SYSTEM/390 computer, all of which are available from the International Business Machines Corporation™, Armonk, New York. User access in this invention is by way of terminals on whose behalf applications execute in the computer. These applications may include an application program with embedded SQL statements, or may comprise query processors.

Please amend the paragraph beginning on page 10, line 17 as follows:

In response to the call made in step 304, the DM in step 306 creates control blocks necessary for the requested operation. The TS_SETUP logic used in this step is illustrated in Table I, where the DM 212 creates a TS control block 402, if one has not been created in response to a previous TS_SETUP call for this access request. The logic also creates one or more TS access control blocks 404, 406, 408 for each statement in the access request. The TS control block 402 is built for the database table that contains the data requested by the application. Certain information from the entry for the table in the database directory 400 is copied ~~copies~~ to the TS control block 402. Such information includes the state of the SPL flag, the selected locking granularity (row or page), and the lockmax value. If the table space is partitioned, and selective partition locking (SPL) is enabled for the defined partitioned table space (i.e. LOCKPART YES in a CREATE or ALTER TABLESPACE statement), the DM 212

checks any predefining conditions for using SPL. For example, SPL would not be available to applications whose access path uses a type 1 index. SPL is also not needed if a statement requires locking the entire table in an exclusive mode. If the conditions do not prevent SPL, the DM 212 sets an SPL control flag 410 in the TS control block 402. For each statement, the DM stores the lock state requested by the RDS in a TS access control block of the statement (for example, at 436 in control block 404). Otherwise, if SPL is not to be used, the entire table space is locked in the highest requested lock state. When TS_SETUP processing is completed, the DM 212 returns to the RDS 210.

Please amend the paragraph beginning on page 11, line 15 as follows:

In step 308, the RDS 210 calls the DM 212 to determine which rows qualify for processing by satisfying the conditions of a statement. At least one call is made for each statement. Each call directs the DM 212 to qualify one row in the identified table according to conditions set forth in the statement. For a statement in which "n" rows are to be qualified, looping occurs in the RDS, with the RDS calling the DM n+1 times. The first n calls return data; the (n+1)th call says "finished". Each statement type requires a call to a particular piece of DM logic. A SELECT statement requires a call to DM_SELECT (Table II); DELETE and UPDATE statements require calls to DM) DELETE/UPDATE (Table III): INSERT statements require calls to DM_INSERT (Table IV). The DM 212 responds in step 310. The first step of each of the DM functions requires the requested lock state to be transferred from the TS access control block for a statement to the TS control block (into field 412). The table is then accessed and its rows are analyzed. For example, in Table II, a SELECT statement causes the DM 212 to check for the rows of the table identified in the statement against the conditions set forth in the statement. For each row that meets the qualifications of the statement, the DM 212 first checks to determine

whether conditions have been met which allow locking at the requested level to be avoided. In step 309, one such condition is whether it can be determined that the data in the rows have been committed. If so, then the partition lock is avoided, i.e., not applied. If not, the DM 212 checks whether the page containing the row has been locked. In this regard, the granularity of low level locks as between row and page is one of the conditions that can be set by a user and stored in the database directory 401. For any statement in an access request in which a qualifying row or a page containing the qualifying row is not already locked, the GETLOCK logic of Table V is called. The GETLOCK logic uses the TS access control block for a statement being processed and the TS control block for the table referenced in order to determine which partition should be locked. A partition to be locked is a partition containing the qualified row, and the partition is locked by calling the LOCKPART logic to lock the partition in the state specified by the RDS; that state is contained in the TS control block 402 in field 412. When LOCKPART returns to the GETLOCK logic, if the partition is locked with an intent lock, the GETLOCK logic then proceeds to lock the qualified row or page at the level of granularity specified in the TS control block 402 in field 416. The number of row or page locks acquired in ~~acquire din~~ the GETLOCK logic for the table is accumulated in the TS control block at 417. If lock escalation is enabled (by a non-zero lockmax value) and the number of row locks (or page locks) acquired is greater than the lockmax value, then the GETLOCK processing calls the LOCKESCA logic of Table VII.

Please amend the paragraph beginning on page 13, line 18 as follows:

In step 312 of Fig. 3, if lock escalation is enabled for the table, the LOCKESCA logic Table VII, when called by the GETLOCK logic, first determines whether SPL has been enabled. When SPL has been enabled, the lock state of each partition of the table being accessed by the application is considered. If a partition is locked with an intent lock (IS, IX, or SIX), that state of

the lock for the partition is upgraded to the appropriate gross lock. In this regard, in the table space, if a partition is locked IS, the lock is upgraded to S; if a partition is locked IX or SIX, the lock is upgraded to X. When a lock is ~~upgrade~~ upgraded, the new lock state for the partition is recorded in the appropriate subfield of the Table control block 402. In addition, an escalation flag (Esc flag) 419 is set in control block 402 by which later requests by the same application for other partitions of the same table will be escalated automatically.

Please amend the paragraph beginning on page 15, line 3 as follows:

The TS access control blocks 404, 406, 408 have essentially identical formats, and only the format of TS access control block 404 will be described. There are typically as many TS access control blocks as there are statements in an access request, and the TS access control blocks of an access request are linked to a TS control block by a pointer. Thus, the TS access control block 404 includes a pointer field 422 containing a pointer to the TS control block 402. In addition, the TS access control block 404 includes a field 424 into which the acquired state of the partition lock for the associated statement received from the RDS 210 is placed; a field 434 denoting the current position of the DM process within the table space in the form of a record identification (RID), which contains the page and partition number where the row is located; and a field 436 containing lock information about the page that contains the just-qualified row.

Please amend the paragraph beginning on page 16, line 20 as follows:

When the application 205 requests access to a record in the database 202, the DBMS 201 will request a lock on the record, or on the page that contains the record, if page level locking is in effect. Before the record/page lock is requested, the DBMS 201 will also request an object lock in the database 202, where the record is stored. In the prior art, the object lock would be on

the table containing the record. The table lock will need to be forwarded to the GLM 222 by RLM 214. The record/page lock requested by DBMS 201 is associated with the table lock. If another DBMS has a lock on the table, the record/page lock will also need to be forwarded to the GLM 222 by RLM 214.

Please amend the paragraph beginning on page 17, line 20 as follows:

This statement is an extension of a known LOCK TABLE statement; it may be used by a programmer to explicitly lock one or more partitions of the named table in either exclusive (X) or in shared (S) mode. Of course when locked in exclusive mode, no other application can access any part of a locked partition for any purpose for as ~~s~~ long as the lock is held. In shared mode, other applications can access a locked partition for read-only access.

Please amend the paragraph beginning on page 18, line 5 as follows:

When the invention is practiced as a procedure performed by a programmed general purpose digital computer, it may also comprise an article of manufacturing, embodied as a computer program product in a form illustrated in Fig. 5. A program comprising a plurality of program steps, or means for performing program steps, written in a computer language, is compiled into object form and stored on a program storage device such as a disk ~~500~~ 502. The program steps implement the pseudocode of Tables I-VII and may be included in the computer program of a data manager in a DBMS. Those skilled in the art will appreciate that the storage device 502 ~~500~~ is merely illustrative of storage media in which a computer program may be fixed, in whole or in part, for greater or shorter periods of time. Such media include, without reservation, disk, tape, ROM, RAM, system nodes, and network nodes.